POO2

Universidad Nacional de Quilmes

Sistema de Estacionamiento Medido

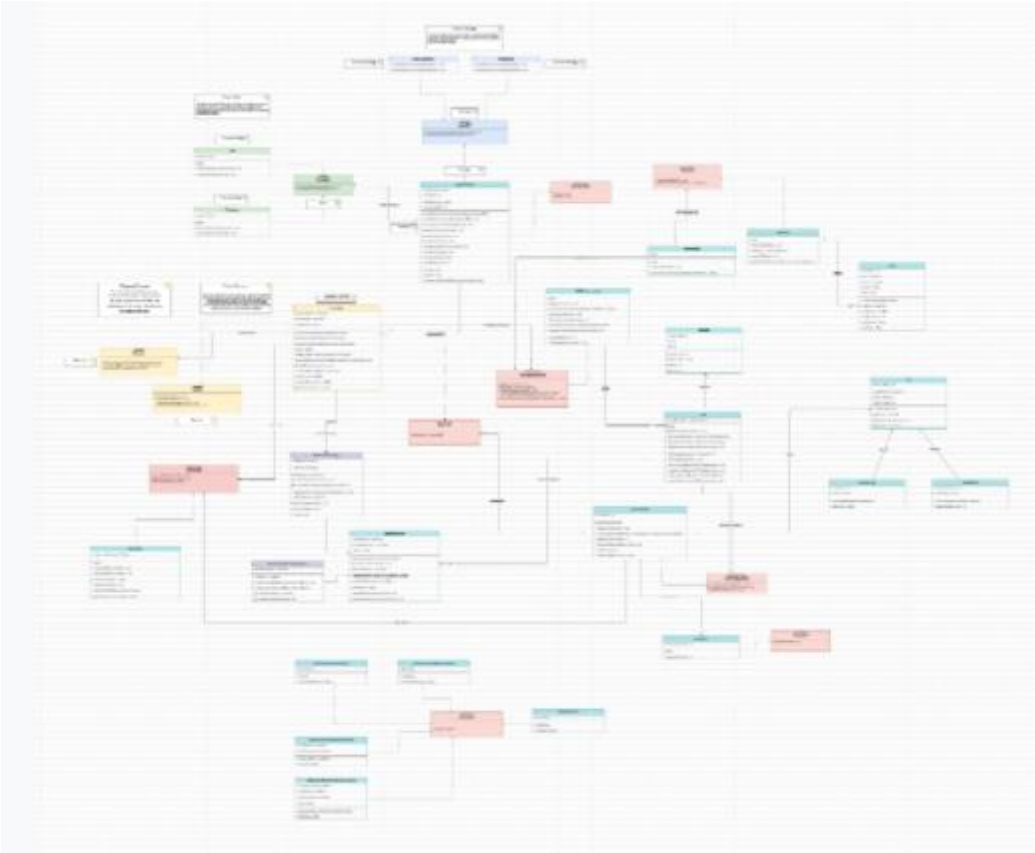
Integrantes:

- Nicolas Nahuel Fernandez – nicolasnahuel.pr@gmail.com
- Federico Plastina – federicogplastina@gmail.com

Testing

Para el testeo de la aplicación se utilizó JUnit 5 y Mockito 3.6.0, habiendo cubierto más del 98% de los casos posibles.

Diseño



Tareas a realizar:

- Diseño de la solución completa utilizando diagrama de clases UML.
- Documentacion en un archivo PDF que incluya los integrandes del grupo y sus direcciones de email, las decisiones de diselo utilizados y los roles segun la definición de Gamma et. al.
- Implementacion completa en lenguaje Java, con test de unidad con cobertura de 95%.
- Alojar el proyecto con acceso público (Github o Gitlab).

Introducción:

A lo largo del proyecto adquirimos diversos conocimientos sobre distintos patrones de diseño y buenas prácticas por lo que estos se incorporaron poco a poco en el desarrollo del proyecto, tanto en código como diseño UML.

Tratamos de mantener en toda la aplicación las buenas prácticas de S.O.L.I.D. Buscando lograr una aplicación más robusta, flexible y mantenible.

Asi mismo aplicamos TDD, asi pudimos construir y verificar el comportamiento de las clases, comprobando diversos casos limite como funcionalidades.

Con el fin de no sobrecargar algunas clases decidimos crear Gestores u Administrador que nos ayudaran a abstraer las responsabilidades. Por ejemplo, GestorSEM, SEMCelular y SEMmulta.

Tambien diseñamos interfaces como IGestorSEM e ISEMCelular a modo de simplificar el protocolo de sus implementadores y poder facilitar la interacción necesaria desde clases externas, por ejemplo Punto de venta.

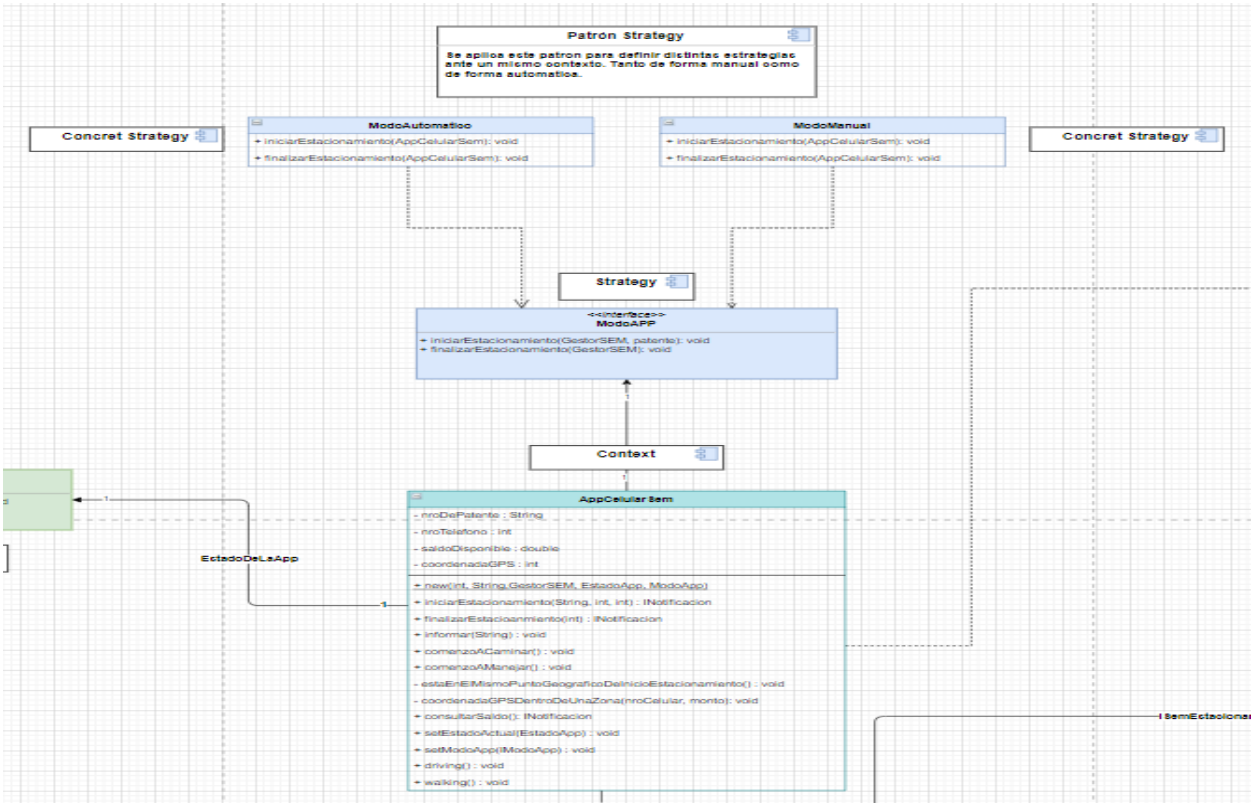
Decidimos que SEMEstacionamiento conozca a sus respectivas Zonas y a su vez estas conozcan a su inspector asignado, puntos de ventas y estacionamientos. Delegando en cada entidad la responsabilidad que le conlleva. Por ejemplo, un una Zona conoce a sus puntos de ventas y estos a su vez registran las ventas hechas

Notamos también un “pasa manos” entre las clases Usuario y AppCelularSEM de responsabilidades en nuestros primeros modelos por lo que decidimos descartar Usuario y concentrar en AppCelularSEM

En el caso de las notificaciones decidimos que estas se puedan utilizar desde distintos puntos de la aplicación. Para esto creamos la interfaz Inotificacion.

Patrones de diseño

Strategy:

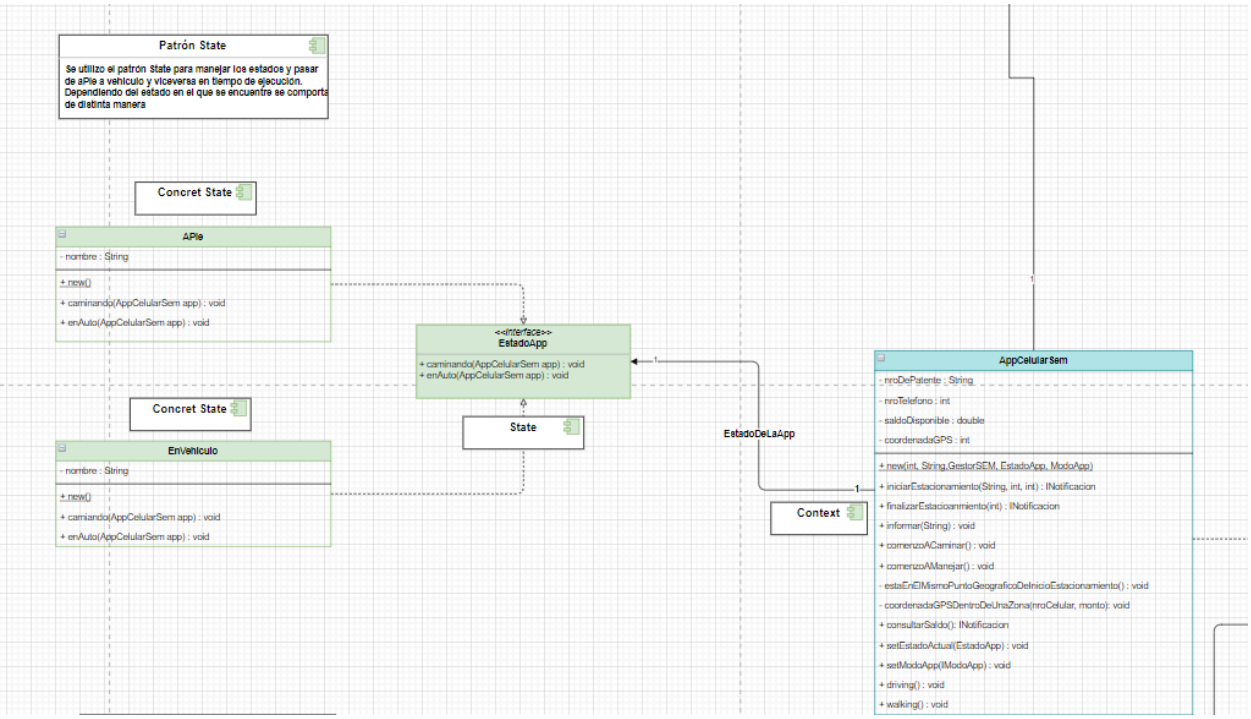


Se aplicó este patrón para cambiar de modo (automático/manual) en tiempo de ejecución cuando el usuario lo desee. Según la estrategia que elija el usuario el algoritmo de Inicio y fin de estacionamiento se comportara de distinta manera.

Roles:

Context -> Class AppCelularSem
Strategy -> Interface ModoApp
Concret Strategy -> Class ModoAutomatico, Class ModoManual

State:

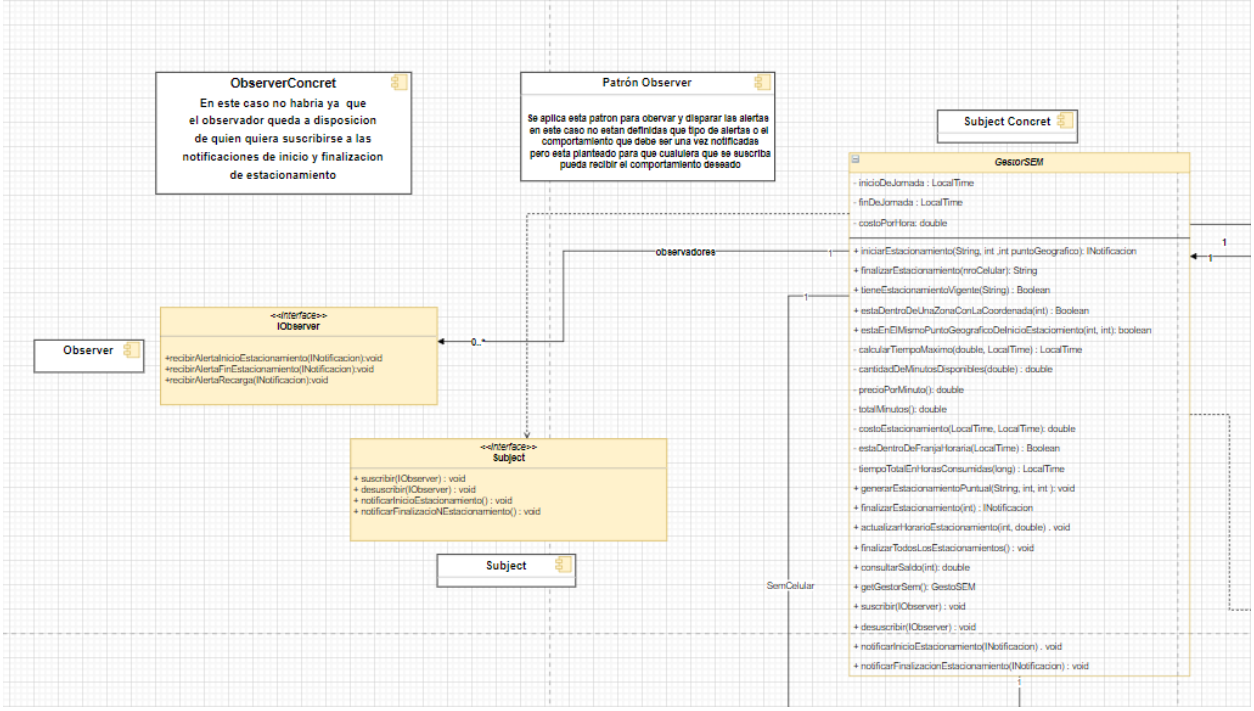


Se aplicó este patrón para cambiar el estado del contexto en tiempo de ejecución, en este caso al notar un cambio de movimiento mediante la interface MovementSensor. Alterando así el comportamiento cuando el estado interno cambia.

Roles:

Context -> Class AppCelularSem
State -> Interface EstadoApp
Concret State -> Class APie, Class EnVehiculo

Observer:



Se aplicó este patrón para enviar notificaciones de Inicio y finalización de estacionamiento en cuanto el usuario dispare los mismos. En este caso por modelo de negocio no hay suscriptores para las alertas, se deja abierta a la suscripción quien desee recibirlas.

Roles:

Observer -> Interface IObserver
Concret observer -> Libre a implementación
Subject -> Interface Subject
Subject concret -> Class GestorSEM